Lab 5: Embedded Systems Testing
Part 1: 10 Points
Due Date: 6/21/2022

Write and test the embedded code to run an electronic device with the following features:

The device will read and display the following:

1. **Current** humidity and temperature levels
2. The **maximum** and **minimum** readings of both humidity and temperature since it was last reset by the user
3. The current **trend** of both humidity and temperature.
    a. The trend of humidity will compare the last two humidity readings and determine if humidity is increasing, decreasing, or stable.
    b. The trend of temperature will compare the last two temperature readings and determine if temperature is increasing, decreasing, or stable.
4. Humidity **Check**: The device will display a humidity check status:
    a. OK if the relative humidity is between 30% and 55%
    b. High: if relative humidity is above 55%
    c. Low: if relative humidity is below 30%
5. The device has a **reset** button to reset the maximum and the minimum values of both humidity and temperature to current readings

Input:

1- The device will read the **current relative humidity** from an input humidity sensor
    Allowed input range: 0% – 100% relative humidity
2- The device will read the **current temperature** from an input temperature sensor
    Allowed input range: 0 – 125 degrees Fahrenheit

Sensitivity of temperature sensor is one degree Fahrenheit
Sensitivity of humidity sensor is 1%

Output:

Your embedded code will display the following values:

For relative humidity:

1. The current relative humidity
2. The maximum relative humidity reading since it was last reset
3. The minimum relative humidity reading since it was last reset
4. The relative humidity trend (up, down, or stable)

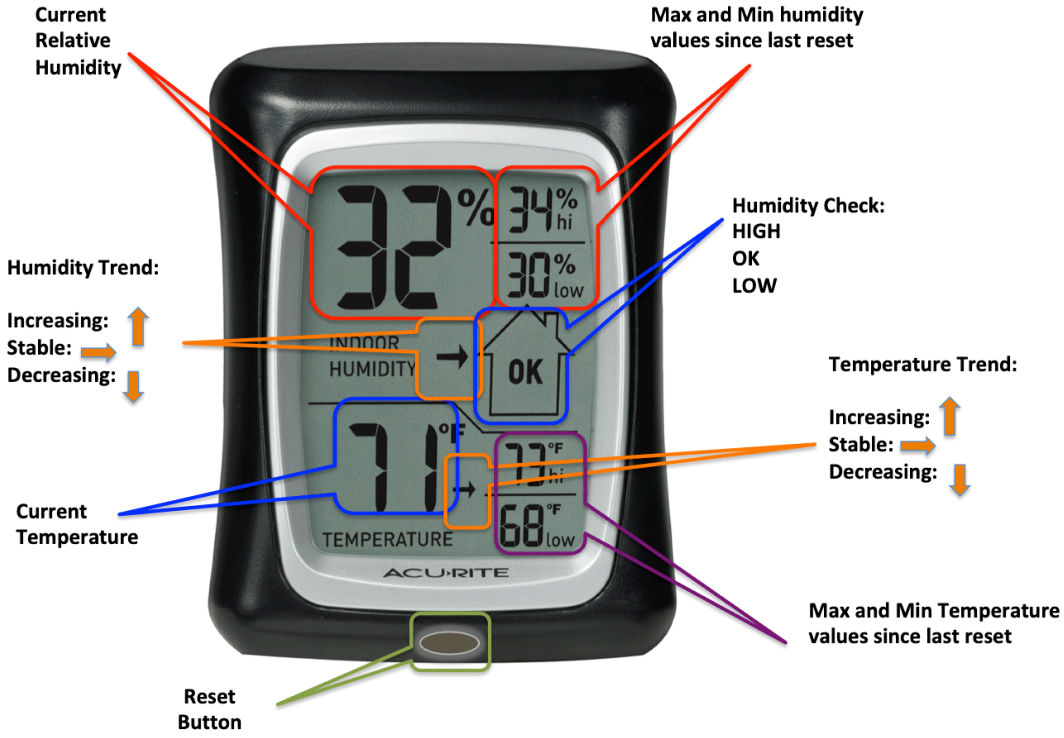
For temperature:

5. The current temperature
6. The maximum temperature reading since it was last reset
7. The minimum temperature reading since it was last reset
8. The temperature trend (up, down, or stable)

For Humidity Check:

9. Humidity status (Hi, OK, or Low)

See next page for demonstration →



Part 1: Deliverables:

1- Submit a working source code to run this embedded system.

   a. For **MANUAL STEPWISE** testing purpose, the **input** can be taken from the keyboard one pair at a time: Since we are not deploying the embedded system in a device yet, the input will be simulated as prompting the user (tester) to input one pair of keyboard values for current temperature and humidity.

   b. The **output** can be simple alphanumeric output values displayed as text; no need for graphics. For example, for humidity the output will be:

      Current Humidity: 33%
      Maximum Humidity: 52%
      Minimum Humidity: 31%
      Humidity Trend: Increasing

   Do the same for temperature and the humidity check

Make sure your code compiles and runs correctly.
Take screenshot of 3 different input and output values and submit them with your source code.

Tips:

- Make sure you initialize your values **correctly** after each reset.
- Think of the Reset as a function or a class,
- Java has multiple time libraries that you could use
- Select your sensor reading (and hence your reading refresh rate) wisely. Reading too frequently (e.g. every second) will deplete the battery fast. Reading every few minutes will provide low-quality feedback (with high latency.)

Lab 5: Embedded Systems Testing
Part 2: 10 Points
Due Date: 6/21/2022

Question a) Design at least **two** test cases (using assertion) to test each of the output values separately.

- Current humidity
- Current temperature
- Max humidity
- Max temperature
- Min humidity
- Min temperature
- Humidity Trend: Write separate test cases for each trend (up, stable, and down). 3 tests total
- Temperature Trend: Write separate test cases for each trend (up, stable, and down). 3 tests total
- Humidity status: (Hi, OK, and Low.) 3 tests total

Hint: To test trend (or max, min), you need to input at least two values before you check the output.

Total number of test types: 15, total number of test cases: 30
Each test will include test code (script) with input values **or input sequences** as needed for each test
Run your tests and take a screenshot of each one of them.
Submit your test scripts

Question b) Data-Driven Testing to avoid test code bloating:
Hint: Review slides 21 & 22 of chapter 3 in the textbook, the "adding two numbers" example", @Parameters

Write the following tests using the following values in two different testing styles:

Style 1: One sequence of seven "Temperature, Humidity" pairs, followed by ONE test at the end of the given sequence (i.e. one test for the entire sequence of seven pairs of input)

Style 2: One pair of "Temperature, Humidity" input followed by a test, repeat seven times (i.e. seven independent tests of one pair of input values).

i. Use the following **temperature** sequence after a reset: **66, 68, 69, 67, 63, 59, 53**
(Use **any** arbitrary value for all the corresponding humidity readings)

Submit your test scripts for both style 1 & style 2
Take a screenshot of the 9 output values after **each test** is run (style 1 & 2) and include them in your deliverable

ii. Use the following **relative humidity sequence** (in %) after a reset: **53, 51, 48, 49, 54, 56, 56**
(Use **any** arbitrary value for the corresponding temperature readings)

Submit your test scripts for both style 1 & style 2
Take a screenshot of the 9 output values after **each test** is run (style 1 & 2) and include them in your deliverable

Question c) Refactoring:
You will notice that the algorithms used for humidity and temperature are similar (for calculating the **max**, the **min**, and the **trend**). Refactor your code (if needed) to use one algorithm that could be used in both humidity and temperature calculations.

Part 2 Deliverables:

i) Submit your test scripts as explained above

ii) Include one screenshot of the 9 output values after each of the two sequences (question a) are read by the system in both styles 1 & 2

iii) Answer the following question: What is the difference between testing the 7 inputs **in a sequence** and testing them **individually**. How are the two test cases **designed**? (use narrative description, no test code needed.)

iv) Submit your refactored code (with comments)

v) As the same person who developed, refactored, and tested the code, does your refactored code make it easier or harder to test it, explain with examples?

vi) If you received the refactored code (written by another developer) to just test it, would it be easier or harder than case iv) above?

vii) Would you prefer to test the original code or the refactored code, if both were written by another developer?